(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
31 December 2003 (31.12.2003)

PCT

(10) International Publication Number
WO 2004/002062 A1

(51) International Patent Classification[7]: H04L 12/24

(21) International Application Number: PCT/EP2002/006975

(22) International Filing Date: 24 June 2002 (24.06.2002)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicants *(for all designated States except US)*: **SIEMENS AKTIENGESELLSCHAFT** [DE/DE]; Wittelsbacherplatz 2, 80333 München (DE). **LABORATORIES FOR INFORMATION TECHNOLOGY** [SG/SG]; 21, Heng Mui Keng Terrace, Singapore 119613 (SG).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **BRAUN, Peter** [DE/DE]; Schwanseestr. 25, 81539 München (DE). **APPAN, Ponnappan** [IN/SG]; Serangoon Ave 4, Blk 229, #10-47, Singapore 550229 (SG). **LINGHIA, Yang** [SG/SG]; Woodlands St 41, Blk 421, #09-173, Singapore 730421 (SG). **PILLAI, Radhakrishna** [IN/IN]; Raghava Vilasathu Veedu, perumpuzha, Quilon 691504, Kerala (IN).

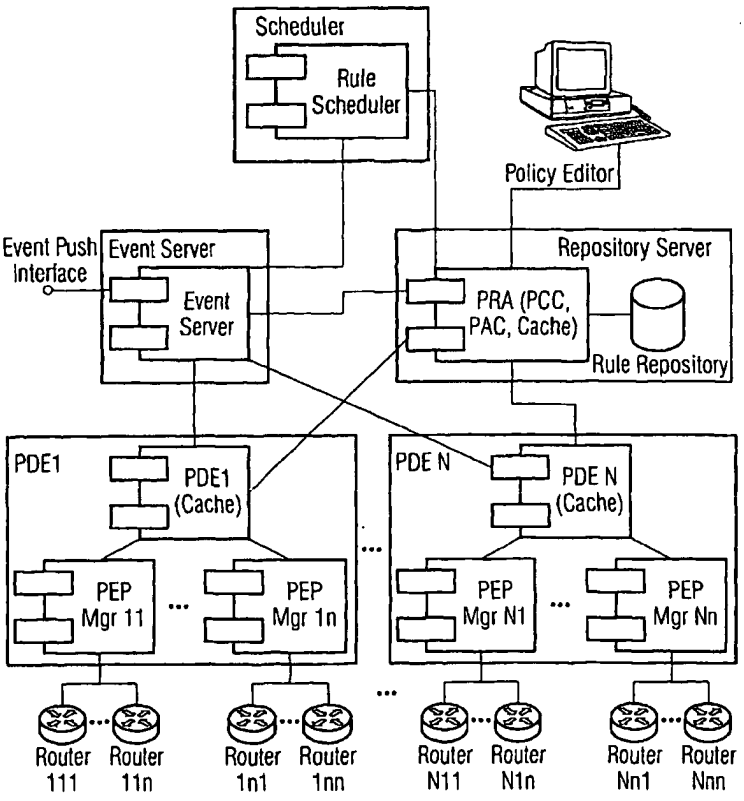
(74) Common Representative: **SIEMENS AKTIENGESELLSCHAFT**; Postfach 22 16 34, 80506 München (DE).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW),

[Continued on next page]

(54) Title: A POLICY MANAGEMENT METHOD AND SYSTEM



(57) Abstract: Policy Management System comprising - a first plurality of event generators, - a second plurality of event consumers, - at least one event server, and - a communication mechanism coupling the event generators with the event consumers via the event server and providing event generation and reception.

WO 2004/002062 A1

Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Description

A Policy Management Method and System

The invention relates to a policy management method for managing the distributed network elements of a computer network as well as to a corresponding system.

1 Introduction

With the increasing complexity, heterogeneity, size of computer networks and the number of supported applications, their management is also getting more complicated and error-prone. To be able to manage the distributed network elements at a central station using human understandable languages and to be able to react dynamically according to the changes in network state are the major challenges for the network management.

Policy Based Network Management (PBNM) is the industrial choice to manage the distributed network elements at a central station using human understandable languages. A typical architecture of policy-managed network is depicted in Figure 1. Policies encode the high-level goals and requirements of management. Policy servers, (identified as PS-[1,2,3] in Figure 1) which are also called Policy Decision Point (PDP), translate the high-level policies into the lower-level device configuration information and then transport them to the network devices which are called Policy Enforcement Points (PEP). Each policy server serves one network administrative domain (identified as Network-[1,2,3] in Figure 1). A policy server can communicate with the adjacent domain policy servers for proper establishment of

the device configuration requirements in the adjacent domains. The policy rules are authored using an editor in the policy management console and stored in the directory server for immediate or later enforcement.

5

However, the PBNM technology is still in its infancy stage. Problems like the detection of inconsistencies or conflicts among the policy rules and the execution of the correct set of policy rule upon a change in the network situation are 10 still not solved yet.

In today's complex, multi-vendor environments, successful, scalable management of network depends on the specification of unified, scalable administered policies. These policies 15 must then map to the configuration of multiple heterogeneous systems, devices, applications, and networks, for the purpose of policy enforcement. The resulting cooperation of these multiple managed entities produces aggregate behavior consistent with the desired policies, and which, in turn, 20 enables the delivery of the desired QoS. Several RFCs and Internet drafts have been published by IETF in the area of PBNM.

The Common Information Model (CIM) [„Common Information 25 Model: Implementing the Object Model for Enterprise Management", Winston Bumpus, John W. Sweitzer, Patrick Thompson, Andrea R. Westerinen, Raymond C. Williams, Wiley, John & Sons, Incorporated] is a collaborative work of the world's leading experts on management technology, information 30 modeling, and information systems architecture. Over the short number of years of its existence it has already achieved broad industry support and adoption. It is the basis of management architectures for the foreseeable future. The

model and its ancillary standards are being developed within the Distributed Management Task Force (DMTF), which is made up of the leading companies in today's computing industry. The CIM describes management information and offers a framework for managing system elements across distributed Systems. CIM is not bound to a particular implementation, and this flexibility allows different systems and applications to exchange and interpret management information.

An object-oriented information model for representing policy information is currently under joint development in the IETF Policy Framework WG and as extensions to the Common Information Model (CIM) activity in the DMTF. This model defines two hierarchies of object classes:

- structural classes representing policy information and control of policies,

- association classes that indicate how instances of the structural classes are related to each other.

The policy classes and associations defined in the model are sufficiently generic to allow them to represent policies related to anything.

Since it is a standard, some of the implementation aspects like the architecture, policy evaluation and consistency checking are intentionally left unspecified.

US 5,889,953 describes a method and apparatus for determining an enforceable policy applicable to one or more network devices. The method includes attaching one or more rule elements to one or more domain elements to create policies,

the domain elements representing network devices and groups of network devices, and the rule elements defining actions, a method for determining whether a conflict exists between the polices, and a method for resolving the conflicts to produce one or more enforceable policies. This patent groups network devices into domain elements. Our invention groups network devices using classes, and class associations, which is a totally different concept.

WO 01/19031 proposes a policy-based network management (PBNM) architecture that can extend network capabilities for a network. The method includes sending a first message from a policy enforcement point (PEP) to a policy decision point (PDP) in response to an external action, and sending a Java object in a second message from the PDP to the PEP in response to receiving the first message. The Java object may be executed on the PEP to implement a policy. Java is the enabling technology for this patent.

WO 00/78004 shows a unified policy management system for an organization including a central policy server and remotely situated policy enforcers. A central database and policy enforcer databases storing policy settings are configured as LDAP databases adhering to a hierarchical object oriented structure. Changes in the policy settings made at the central policy server are automatically transferred to the policy enforcers for updating their respective databases. Each policy enforcer collects and transmits health and status information in a predefined log format and transmits it to the policy server for efficient monitoring by the policy server. For further efficiencies, the policy enforcement functionalities of the policy enforcers are effectively partitioned so as to be readily implemented in hardware. In

addition, policy server and policy enforcers may be configured for high availability by maintaining a backup unit in addition to a primary unit. The backup unit becomes active upon failure of the primary unit.

US 6,301,613 describes a method and apparatus which are provided for verifying policies that govern a policy-based system. The method and apparatus may be implemented as a policy verifier that acts upon one or more policies. Each policy comprises a condition and a consequent. The policy verifier acquires configuration information about the system under management, thereby acquiring an understanding of the system. The policy verifier determines whether all the policies can be satisfied by the system, and if not, reports problems or errors in the policies that cause the policies to be non-satisfiable. The policy verifier determines whether all the policies are feasible for the system, and if not, reports problems or errors that cause the policies to be non-feasible. The policy verifier also verifies that a configuration required by a particular policy consequent can be actually carried out by the managed system. In one embodiment, the policy verifier operates on network management policies, of a policy-based network management system. As a result, this document improves the accuracy and safety of policies prepared for a network that previously did not use policy-based management.

It is an object of the invention to improve the prior art policy management methods and systems, respectively, in particular with respect to the flexibility and general applicability thereof.

This object is solved with a system according to claim 1 and

a method according to claim 15, 22 or 24, respectively.

With the present invention, an event based distributed policy network management system for an automated and consistent management of networks is presented. The managed network elements are modeled using object oriented approach. They are abstracted into a group of classes, called schema. The classes are related to each other by one of the following relationships: inheritance, associations or aggregation. The managed network elements are instance of the classes.

Event server is used for distributing events among components in the system. The events must be constructed using the classes and the classes' properties defined in the schema. Components that have something need to notify other components construct events and forward the events to Event server. Event server is responsible for the delivering of events to the listening components according to their interests. Each component reacts on the event they received according to the event attributes.

The invention is described in detail below, discussing basic aspects and preferred embodiments thereof on the basis of the drawing in which

Fig. 1 shows the network architecture of a general PBNM system,

Fig. 2 a component diagram of an event driven distributed PBNM system,

Fig. 3 a deployment diagram of an event driven distributed

PBNM system,

Fig. 4 a policy creation scenario which is adapted to a system according to Fig. 1 and 2,

Fig. 5 a flow diagram for policy creation and modification for PCC,

Fig. 6 a simple network typology in an embodiment of the invented system,

Fig. 7 a flow diagram showing the procedure of PAC for the case of lock with no weight,

Fig. 8 the flow diagram for a procedure of PAC for the case of lock with weight.

Section 2 gives an overview on all the components that are part of the event based distributed policy network management system. Section 3 illustrates a process that determines the related policy rules from the received events. Section 4 presents an apparatus for checking the conflicts among policy rules and the applicability of the rule with the help of schema and network element information. It also shows a method for coordinating access of policy rules amongst a plurality of processes in a distributed system.

The main novel features or advantages, respectively, of a preferred event based policy management system according to the invention are:

- Generic Policy Decision Point architecture that is domain independent,

- a policy rule retrieval and enforcement process that is driven by events dynamically generated by the managed objects,
- a way to relate the policy rules to the events using a event structure invented,
- a caching mechanism to cache the policy rules,
- a Policy Rule Consistency checking mechanism fro checking of inconsistency and conflicts using the policy rules, the management information model and managed object instances,
- Policy Repository Adapter that manages the read and write access to an IETF compliant policy repository including a Policy Consistency Checker and an efficient Policy Access Controller,
- distributed and scalable architecture that ensures the consistency between Policy Repository and PDE even in a distributed environment with more than one policy Server.

2 Event Driven Distributed Policy Network Management System

An Event Driven Distributed Policy Network Management System (EDDPNMS) manages network domains that comprise of one or more PEPs. Figure 2 shows the component diagram of the EDDPNMS architecture. The EDDPNMS comprises of the following components:

Event Server (ES) - which is involved in the distribution of events between components. Each component can publish the event types that it is capable of generating and other components can subscribe to the events that they are interested in. Event server acts as a forwarder of events between the set of components acting in event generator role to the set of components acting as event consumers.

Policy Repository Adapter - is used for the storage and retrieval of the policy rules from the policy repository. This acts as event generator of policy rule create/delete/modify events. It has the following sub-components:

Policy consistency checker - checks for the consistency of a newly created policy rule with the existing rules.

Policy Access Controller - manages the concurrent access to the policy repository by different clients.

PEP Manager - is in charge of managing one or more PEPs. It maintains and monitors the Status of each PEP. It maintains the list of capabilities of each PEP device and converts the device-independent requests for creation/deletion/modification of management objects to device-specific commands. It communicates with the PEPs using one of the standard management protocols like Simple Network Management Protocol (SNMP) or Common Open Policy Service (COPS).

Policy Decision Engine - is responsible for the evaluation and execution of the policy rules.

Policy Rule Cache - stores the sets of policy rules that have recently been retrieved by PDEs to reduce the policy rules retrieval time. A hierarchical distributed caching mechanism should be implemented; caches should be placed in Policy Repository Adapter (PRA) and Policy Decision Engines (PDE). The goal of caching is to reduce the time taken for policy decisions. The time taken to retrieve the policy rules from the rule repository (e.g. from a LDAP server) contributes the

most to the delay in the rule retrieval process. When there are multiple PDEs, it is more efficient to place a policy rule cache component in the PRA. As every event generated might result in multiple PDEs to retrieve the same set of rules, with the cache in the PRA, the number of access to the rule repository can be reduced to one. The PDEs make decisions based on the related policy rules, the information model and states of the object instances. To avoid the frequently access of the rules in PRA, a local cache for policy rules is required.

Rule Scheduler - responsible to trigger events to notify the installation or de-installation of the policy rules with time period condition. The scheduler can be either a standalone component or part of the PDE and/or PRA. The benefit of being a standalone component is that the scheduler can perform other tasks than rule scheduling, e.g. housekeeping task like backing up of database, used as an alarm clock by other components for periodic tasks. However, this solution increases network traffic and it is not accurate enough for time critical tasks. If the scheduler is a subcomponent of PRA, this will reduce the network traffic, however, it loses the flexibility of serving for other purposes at the same time. On the other hand, if the scheduler is part of PDE, it can reduce network traffic and network delay. This approach is ideal for time critical policy rules. Its disadvantage is that there may be several schedulers performing scheduling for the same policy rules at the same time, apart from the loss of flexibility. Another aspect to improve the performance of the rule scheduler is to register the rules only when they are retrieved by an event. Instead of scheduling all the rules with time condition, only those rules that required by the system are scheduled. This can

reduce the system load and improve scalability if there are a lot of policy rules with time period condition.

The event driven architecture makes the design extensible and flexible for adding policy based management of new domains. It provides a uniform interface for interfacing with PEP manager. It is also efficient since the policy rules are processed only at the reception of events. Figure 3 shows a possible deployment of the proposed architecture. The components communicate with each other using Middleware. Existing technologies like CORBA/IIOP, XML/SOAP, Java RMI, COM or Message Oriented Middleware (MOM) can be used for this.

3 Policy Rules Processing Using Events

3.1 Managed Element Schema

The elements managed by the EDDPNMS are depicted using a schema. The schema provides the actual model descriptions along with a set of classes with properties and associations that establish a well understood conceptual framework, within which it is possible to organize the available information about the managed environment. One of the possible technologies to be used is a CIM Schema.

Classes can contain properties, which describe the data of the class, methods, which describe the behavior of the class, and constraints, which describe the rules that the instances of the class have to follow. They define the basic unit of management. Each class is a template for a type of managed object; all instances of the type use the template. For example, the class called NetworkInterface represents

all the network interfaces managed by the system. It can have properties like: hardware bandwidth and diffserv bandwidth; method like setDiffServBandWidth; and constraints like: hardware bandwidth >= diffserv bandwidth.

Classes are related to each other by the following means: inheritance, associations or aggregation. Associations are represented using classes too.

3.2 Definition of Policy Rules

Each policy rule consists of a conditions part and an actions part. The conditions part specifies the conditions under which the actions have to be executed. The conditions part is made up of one or more simple conditions combined by logical operators and each simple condition contains a variable name and a value related by a relational operator.

3.3 Event Types and Format

Evaluation of policy rules is triggered by internal or external events (e.g. from network devices or from internal time scheduling events which occur due to scheduling of policy rules). The type of the events can be classified into the following categories:

- Object Creation: Reports the creation of an object within the management system. The object can be a new user, a new flow or a new network device, etc.

- Object Deletion: Reports the deletion of an object within the management system. The object can be a logout user, a terminated flow or a faulty network device, etc.

- State Changed: Reports the changes of state of an object within the management system. It can be a change of priority of user, an enabling of policy rule, or a network device changed to standby state.

- Attribute value changed: Reports the changes in the attribute of an object. An example is a network device with a change of current bandwidth usage.

- Threshold crossed: Reports the threshold crossing events happened in the objects in the management system. Some of the examples are congestion in a link or hard disk space available for the repository is below a safe level.

Each event has a field to defined which category it belongs to and a list of attributes which describe the event in more detail. The following are some examples of events with their attributes:

1. PEP startup event with the PEP ID and PEP device inventory information (one manifestation of this is a COPS-PR REQ message that is sent with the interface role combination information). It belongs to object creation event.

2. Link bandwidth threshold crossing event (Performance related). It belongs to threshold crossed event.

3. PEP device operational status events (one example is a COPS RPT or SNMP TRAF message that is generated by the PEP for removal or insertion of a new device) It belongs to state changed event.

4. PEP access event by a particular user using a particular application (one example is an RSVP outsourcing message request while the bandwidth reservation for a new flow is done). It belongs to object creation event.

5. PEP de-access event (one example is an RSVP outsourcing message request while the bandwidth reservation for a flow is torn down) It belongs to object deletion event.

6. Create/Modify/Delete events for a policy rule from the policy editor. Depends on the operation, it can be object creation, or attribute value changed or object deletion event.

7. Events from an adjacent domain PS. (one example is a PEP removal event that affects the adjacent domain PSs or a modification of SLA).

8. User login event. This allows the PS to provision user specific configurations. It belongs to object creation event.

For example, the events in the System are defined using the following format:

The eventCategory specifies category the event belongs to. The objectClassName states which class does the object instance belong to. The oid is the object instance's ODD. The attribute list contains a list of attributes that defines the event. For every attribute item, qpVariableName

corresponds to the attributes of the classes in the schema which can be made unique by specifying the full path like: <schemaName>. <className>. <attributeName>; qpValueTypes specifies the type of the value associate with the variable; qpOperator defines the relation between a variable and a value. It can be MATCH, GREATER, LESS THAN and so on. Finally, the qpValue field contains the value that characterizes the attribute.

The variable name for the policy rule condition should be one of the properties defined in the management element schema.

To further depict the events, the event generators should attach a list of attributes that can help the PDE to make a decision. E.g. for an event with attribute eventCategory = "object creation" and objectClassName = "Policy Rule", the event generators should also send information on the role combination of the created rule and the conditions of the rule, so that PDE can decide whether to retrieve the rule. This can improve the efficiency of the PDE. The event generators could also provide the policy rule domain as an attribute item in the AttributeList. This would speed up the policy rule retrieval process as it narrows down the scope of search space. The disadvantage of this method is that it reduces the extensibility and maintainability of the system. The event generators have to have the knowledge of how the policy rules are arranged in the rule repository and how the events are related to the type of policy rules, changes in the policy rule organization have to be reflected in the event generators as well.

3.4 Types of policy rules and future extension

Policy rules could be classified as shown below:

- User or application related policies. These are to be applied when resource requests happen (e.g. a new traffic flow is initiated or terminated). Example events are 4 and 5 identified above.

- What are the resources the user is allowed to use?

- What are the applications the user is allowed to use?

- What are the resources the application is allowed to use?

- Device (PEP) policies. These are applied when a new PEP Starts up or is shutdown (example event is 1 identified above).

- How the PEP resources are used (e.g. how the link bandwidth is partitioned among the traffic classes?).

- Fault or performance related policies. These are applied when a fault or performance related events happens in the network (example events are 2 & 3 identified above).

- Inter-Network policies. These are applied when traffic flows across network boundaries.

- What is the SLA between the networks?

- Purely time-based policies - could be used, for example, for setting up a video-conferencing call during certain period of time.

- The conditions of policy rules can specify a time period

during which the policy rule is valid.

- Policy rules that use a combination of user/application/device/network (combinational policies) could also be formed.

The types of policy rules can be easily extended by defining a new policy rule format using schema and adding the rules belonging to the new type into a new branch. The PDE also need to be extended to understand how to evaluate and enforce the new type of rules.

3.5 Event Processing in PDE

The policy decision engine (PDE) listens to significant events that are required for policy rule processing. The event server can be combined with the PDE when there is only one PDE in the system, since the PDE is sole listener for the event server. This can reduce some network delay at the cost of reduced flexibility. When a event occurs, PDE does the following:

1. Analyze the event, check whether policy rule retrieval is required. E.g., if the eventCategory= "object deletion" and objectClassName= "Policy Rule", there is no need to retrieve the rule. If the eventCategory= "object creation", objectClassName= "Policy Rule", if there are installed rules that intersect with the attribute list came with the event, the rule should be retrieved.

2. If the retrieval of rules is required, retrieve the policy rules that are relevant for the event. The relevance of the policy rules can be determined from the event's

attribute list. The event attribute variables in the attribute list should satisfy the conditions in the policy rules or the corresponding attributes of the policy rule.

3. For rules with time period specified as one of the conditions and it is not currently valid, store the policy rule for later processing. The rule will be registered with the scheduler and the scheduler will send out a notification event when the time period condition is satisfied, and another notification when the time period has expired.

4. For the rest of the matching policy rules, converts the policy rule conditions and actions into object class instances - this step is specific to the type of policy rule that has been retrieved.

5. Execute the actions specified in the policy rule in the specified order or in a random order if not specified. The PEP manager can be Multi-threaded for parallel installation of the PRs to multiple PEPs.

6. If the execution of the action is successful, create a policy rule installation information with the policy rule name and the device ID indicating where it is installed. One way to improve the operation is to cache the enforced policies in the PS and in local PS at PEP if one is available. Either the cached time, or the number of cached entries, or both could be controlled.

7. Report the outcome of the policy rule execution to a central logger. One of the embodiment can be to provide a status report to the creator of the policy on the status of enforcement.

Using the event structure defined above, the performance of the event processing can be enhanced by mapping policy rules to event categories. The components that generate events specified the corresponding eventCategory for the event and the set of attributes; the PDE can restrict the search scope only to the rules that has same eventCategory, and retrieves and enforces the matching rules accordingly. The event handling process is very generic, it can handle any kind of events so long as the PDE knows how to enforce the retrieved rules.

Enforceability of a Policy Rule

- A PR might not be immediately enforceable because of time constraints.

- A PR might not be enforceable at a particular device because of the conflict between the capability of the device and the actions expressed in the policy rule. This kind of inconsistency are NOT analyzed by the policy consistency checker in the rule repository adapter but has to be done by the PEP manager before installing the configuration objects at the PEP.

3.6 Sample event processing scenarios:

Event 1: An event is received by the PDE when a new policy rule is created

Procedure: The event with eventCategory = "ObjectCreation", objectClassName = "PolicyRule" and oid= "XYZ" has the following attribute list:

Attribute 1:

qpVariableName = "roles"

qpOperator=MATCH

qpValue= "edge&&diffserv"

1. Analyze the event, check whether policy rule retrieval is required. For this event, the eventCategory= "object creation", objectClassName= "Policy Rule", so we need to retrieve the rule. Check whether any installed rules has the same roles. If there are such rules, the rule needs to be retrieved.

2. Retrieve the rule with oid = "XYZ".

3. If the rule with time period specified as one of the conditions and it is not currently valid, store the policy rule for later processing.

4. Otherwise, converts the policy rule conditions and actions into object class instances - this step is specific to the type of policy rule that has been retrieved.

5. Executes the actions specified in the policy rule in the specified order or in a random order if not specified.

6. If the execution of the action is successful, creates policy rule installation information with the policy rule name and the PEP manager name indicating where it is installed.

7. Also reports the outcome of the policy rule execution to a central logger.

Event 2: Device Startup Event with the link type information

Procedure: The event with eventCategory = "ObjectCreation", objectClassName = "NetworkInterface" and oid = "XYZ" has the following attribute list:

Attribute 1:

qpVariableName = roles"

qpOperator = MATCH

qpValue = "edge&&intserv"

1. Analyze the event, check whether there are rules retrieved using the same event exist in the policy rule cache. If there are such rules, go to step 3. Otherwise the rules need to be retrieved.

2. Retrieve the policy rule with roles = "edge&&intserv".

3. For rules with time period specified as one of the conditions and it is not currently valid, store the policy rule for later processing. Scheduler in the Repository Adapter will send out a notification event when the time period condition is satisfied, and another notification when the time period has expired.

4. For the rest of the matching policy rules, converts the policy rule conditions and actions into object class instances - this step is specific to the type of policy rule that has been retrieved.

5. Execute the actions specified in the policy rule in the specified order or in a random order if not specified.

6. If the execution of the action is successful, create

policy rule installation information with the policy rule name and the device ID indicating where it is installed. One of the possible improvements is to send the differences in the currently installed device object instances and the new object instances to the PEP. This is possible only with SNMP (since COPS does not provide for individual addressing of object instance attributes).

7. Report the outcome of the policy rule execution to a central logger.

Event 3: User-flow access event

Procedure: The event with eventCategory = "ObjectCreation", objectClassName = "RSVPFlow" and oid = "XYZ" has the following attribute list:

Attribute 1:

qpVariableName = "ApplicationID"

qpOperator=MATCH

qpValue= "Video"

1. Analyzes the event, check whether there are rules retrieved using the same event exist in the policy rule cache. If there are such rules, go to step 3., otherwise the rule needs to be retrieved.

2. Retrieves the policy rule with a condition that contains ApplicationID = "Video".

3. For rules with time period specified as one of the conditions and it is not currently valid, store the policy rule for later processing. Scheduler in the Repository Adapter will send out a notification event when the time

period condition is satisfied, and another notification when the time period has expired.

4. For the rest of the matching policy rules, converts the policy rule conditions and actions into object class instances - this step is specific to the type of policy rule that has been retrieved.

5. Execute the actions specified in the policy rule in the specified order or in a random order if not specified.

6. If the execution of the action is successful, create policy rule Installation information with the policy rule name and the device ID indicating where it is installed.

7. Report the outcome of the policy rule execution to a central logger.

7 Policy Repository Adapter in Policy Server

The Policy Repository Adapter (PRA) is responsible for the management of the policy rules. Apart from the subcomponent that interacts with the directory or database, the PRA also consists of two sub-components: the Policy Consistency Checker (PCC) and the Policy Access Controller (PAC). The PCC is responsible for detecting the policy conflict and consistency violation. The PAC is used to make sure that the policy rule can only be modified when nobody else is reading or modifying it, and can only be read when no one is trying to write on it.

The sequence diagram in Figure 4 shows the interaction between various components under normal condition when creating a new policy. It is assumed that the policy is

created to realize a new business goal and supersedes all the prevailing policies. Therefore, the creation of this policy could lead to deletion or modification of some of the existing policies.

1. Policy Editor sends the policy rule that the user is going to create to PRA by calling operation **createPolicyRule()**.

2. PRA sends the policy rule to PCC for conflict checking by calling operation **checkConflict()**. Caching at the PCC can speed up the consistency checking algorithm.

3. PCC calls the **getPolicy()** of the PRA to retrieve all policies stored in directory if its policy rule cache is empty.

4. PCC uses Operation **findRelatedPolicies()** to get all the related policies for the requesting rule.

5. PCC issues read locks for all the related policies by calling the **lockRelatedPolicies()** of the PAC. The Policy Editor should provide a StatusMonitor object to the PRA. So that the PRA can release the all the Locks associate with the Policy Editor in the case of policy editor crash.

6. PCC checks for the conflicting policies and finds out the possible solutions using operation **determinePossibleSolutions()** In the case of policies that needs to be modified the policy consistency checker also provides a range of parameters that is consistent with the policy created. For rules that are not conflict with the requesting rule, the locks issued on them will be released.

7. It is also assumed that the user of the policy management has to approve the deletion/modification of existing policies. Therefore FCC returns the list of possible solutions to PRA. PRA returns the list to policy editor.

8. Policy editor displays the list of possible solution to the user. The user has to adopt one of the solutions before he/she is able to create the policy rule. Policy Editor calls the **updatePolicyRules()** of PRA to update all the policy rules modified in order to adopt the suggestion.

9. PRA calls **checkConformity()** of PCC to check for whether the client's solution conforms with the suggestion it made before.

10. If there is no conflicts returned by the PCC, the PRA updates all the policies accordingly using **updatePolicyRules()** and unlock all the locks associate with this creation request using **unlockRelatedPolicies()** of PAC.

11. PRA notifies PDE about the creation of the new policy and the modification and the deletion of policies, if any by using the **policyRuleEventPush()** Operation of PDE.

The sequence of interaction among Policy Editor, PRA, PCC, PAC and PDE is the same for policy rule creation, modification and deletion. The only difference is the way that the PCC checks for conflicts.

4.1 Policy Consistency Checker

This section presents a Policy Rule Consistency checking

mechanism for checking of consistency and conflicts using the existing policy rules, the management information model and managed object instances. When a policy rule is created or modified, it will first be checked by the general conflict checking process which checks the conflicts based on existing rules. If the rule is able to pass the first level of checking, it will be further checked using schema and the managed object instance information. The following two sections describe the two level of checking accordingly.

4.1.1 General conflict checking

Figure 5 shows how the FCC checks for conflicts in the case of creation or modification of policies:

When a client creates a policy, the FCC retrieves all the related rules. Rules that are related to the requesting rule (RR) if they have the same, subset or superset of key attributes that are used in RR.

E.g. Role is a key attribute that scopes the rule applicability domain. Rule 1 has the role "Edge + DiffServ", rule 2 has the role "DiffServ", rule 3 has the role "Edge + IntServ".

- The role of rule 1 is a superset of the role of rule 2.

- The role of rule 2 is a subset of the role of rule 1.

- The role "IntServ" of rule 3 is mutually exclusive with the role "DiffServ" of rule 1 and rule 2.

The related rules are further filtered according to their conditions. Rules with the following properties have to be further investigated:

(Condition intersect with the RR 's condition) and (TimePeriodCondition intersect with the RR 's TimePeriodCondition)

For all the rules that need further investigation: if its action is in conflict with the action of the RR and its priority is higher than the priority of the RR, insert the conflict into a conflict list with suggested solution. E.g. the network administrator can change the priority of the RR to be higher than the conflicting rule.

An example

The above algorithm can detect conflict in both Intserv and DiffServ rules. E.g. there is one existing rule with the a role = "User + IntServ" and priority = 9

Rule 1: If user is member-of "elite-group", allow guaranteed Service reservations from the user, with MAX BW = 10Mbps. The RR is:

If user name = "John", allow guaranteed service reservations from the user, with MAX BW = 20Mbps. Role = "User + IntServ" and priority = 12

If John belongs to the "elite-group" group, then there is a conflict between these two rules according to the algorithm. That is the MAX BW of the RR is different from rule 1. Therefore the network administrator will be suggested to change the priority of the RR to be less than 9.

4.1.2 Conflict checking with the help of a network topology

The above algorithm can only detect the conflicts among the rules if they have related conditions. It can't detect the conflicts among the rules that are not related by conditions. An algorithm is proposed to detect conflicts among rules with the help of Schema and network information.

1. The PDP obtains the information of managed elements and instantiates to object instances using the network information and the schema.

2. When a rule has passed the general conflict checking, the rule will be sent to PDP for further checking.

3. The PDP simulates the execution of the rule and its related rules, and checks whether the execution violates the constraints specified in the schema.

The example below illustrates how the algorithm works. To simplify it, only the checking of the bandwidth conflicts is shown. The network information includes how the interfaces are interconnected, the egress BW of each interface, the role and the IP address. Take the example of CR in Figure 6. It has the following properties:

- Interconnects with interface eth0: 10.27.0.254 of edge3 through interface eth0: 10.27.0.1.

- Interconnects with interface eth0: 10.24.0.254 of edge1 through interface eth1: 10:24.0.1.

- Interconnects with interface eth0: 10.25.0.254 of edge2 through interface eth2:10:25.0.1

- The maximum egress BW of interface eth0:10.27.0.1 is 150 Mbps.

- The maximum egress BW of interface eth1: 10.24.0. 1 is 150 Mbps.

- The maximum egress BW of interface eth2:10.25.0.1 is 150 Mbps.

- All the interfaces have the role of Edge + DiffServ

When a PDE Starts up, it retrieves the network information of each element that belongs to its manage domain and constructs them as interface objects. Initially each interface object should have the following information:

- IP Address
- Egress BW
- DiffServ BW
- Role
- DiffServQueueList
- OtherQueueList
- ReserveList

And the PDP also constructs the following 3 association objects:

ManagedDevice: It consists the list of interfaces in the same device. It contains information about the device and the list of OIDs of the interfaces in the device.

CII: The immediate interface in other device that sends packets to this interface i.e. connected input interface (CII). It contains the referred interface's OID and the list

of the associated OIDs. It also has the following constraint: the sum of the diffserv bandwidth of the associated interfaces is less than or equals to the bandwidth of the referred the interface.

COI: The immediate interfaces in other device that receive packets from this interface to the destination, i.e. connected output interface (COI). It contains the referred interface's OID and the list of the associated OIDs. It also has the following constraint: the sum of the diffserv bandwidth of the associated interfaces is greater than or equals to the bandwidth of referred the interface.

For example, PDE have created an interface object for one of the interface in CR as following

1. IPAddress = 10.24.0.1
2. Egress BW = ISOMbps
3. DiffServBW = ISOMbps
4. Role = "Edge + DiffServ"
5. DiffServQueueList = empty
6. OtherQueueList = empty
7. ReserveList=empty

Constraint:

1. DiffServ BW + OtherQueueList.bandwidth<= Egress BW
2. DiffServQueueListbandwidth <= DiffServ BW

ManagedDevice:

Name= "CR"

Location= "Lab 1"

OID1 = "CR1.1" // Interface1 = "10.27.0.1"

OID2 = "CR1.2" // Interface2 = "10.25.0.1"

OID3 = "CR1.3" // Interface3 = "10.24.0.1"

CII:

MyOID = "CR1.3" // Interface3 = "10.24.0.1"

OID1= "Edge3.1" // Interface1= "10.27.0.254"

OID2= "Edge1.1" // Interface2= "10.25.0.254"

Constraint:

1. OID 1. DiffServ BW + OID2. DiffServ BW <= MyOID. DiffServ BW

2. For all the related object instances, every DiffServQueue in the DiffServQueueList must have a corresponding DiffServQueue in MyOID. DiffServQueueList or MyOID. ReserveList, and the sum of the corresponding DiffServQueue bandwidth should be less than or equal to the bandwidth of the corresponding queue in MyOID.

3. The same constraint applies to the OtherQueueList.

COI:

MyOID = "CR1.3" // Interface3= "10.24.0.1"

OID1= "Edge1.2" //Interface1="10.0.17.1"

Constraint:

1. OID1. DiffServ BW + OID2. DiffServ BW >= MyOID. DiffServ BW

2. For all the related object instances, every DiffServQueue in the DiffServQueueList must have a corresponding DiffServQueue in MyOFD. DiffServQueueList or MyOID. ReserveList, and the sum of the corresponding DiffServQueue bandwidth should be greater than or equal to the bandwidth

of the corresponding queue in MyOID.

3. The same constraint applies to the OtherQueueList.

After created the object instances, the PDE evaluates the new or modified rules. In order to illustrate the algorithm, let's assume that this PDE is newly setup and all the policy rules are imported from somewhere else. The following rules are arranged according to their priorities, that is the following rules will be executed in sequence.

1. Reserve a 70 Mbps queue in edge router for diffserv traffic.

2. Reserve a 10 Mbps queue in edge router for traffic going to a mission critical machine A and set the DSCP value to EF.

3. Out of the 70 Mbps diffserv queue in the edge router, 30 Mbps will be used for Best Effort Traffic.

4. Out of the 70 Mbps diffserv queue in the edge router, 15 Mbps will be used for Assured Forwarding Traffic.

5. Out of the 70 Mbps diffserv queue in the edge router, 30 Mbps will be used for pre-marked Expedited Forwarding Traffic.

Each managed object instance does the following when a policy rule is executed.

1. If the policy rule's condition is not targeting for it, do nothing.

2. If the execution of the rule alters some of its properties, check whether it violates the constraints specified in the schema. If there are violations, the rule can't be implemented, report the detail to PDE.

3. If there is no violation in the object instance, check with the association classes that related to it. The association classes check for the constraints as well. If there are violations, the rule can't be implemented, report the detail to PDE.

Using the algorithm, the sample rules are walked through as follows:

Rule 1 results in all interface objects with their role = "Edge + DiffServ" Updates their DiffServ BW to 70Mbps.

Before execute Rule 2, PDE instructs all managed object instances with their role = "Edge + DiffServ" simulates the policy action. E.g. Interface 10.24.0.254 has to create a EF queue with BW=10 Mbps. The constraint 1 and 2 are not violated, the association object instances are checked. In the COI object instance for interface 10.24.0.254, its related Object instances 10.27.0.1 and 10.25.0.1 do not have EF queue yet, a queue with BW=10 Mbps/2 is added into the ReserveList, no constraint is found. In the CII object instance, the related object instance 10.0.17.9 doesn't have EF queue created either, a queue is added into the ReserveList, no constraint is found.

The same process applied for rule 3, rule 4 and rule 5. For rule 5, a conflict will be detected as the total number of DiffServ BW in the Edge+Intserv interface will exceed the

70Mbps defined by rules with higher priority. The user will be informed.

Let's assume the BW in rule 5 has been modified to 25Mbps, and rule 5 went through successfully.

The PDE finds that after all the rules are executed, the CR interfaces still have some reservation pending, all the related rules will be notified to the user and suggest that all the CR interface should have at least 35Mbps for EF queues, 15Mbps for AF queues and 30Mbps for BE queues. And three rules will be formed automatically.

- In the Core router, (30 + 23) Mbps will be used for Best Effort Traffic.

- In the Core router, (15 + 23) Mbps will be used for Assured Forwarding Traffic.

- In the Core router, (35 + 24) Mbps will be used for pre-marked Expedited Forwarding Traffic.

The 23 and 24 are derived by the (maximum egress BW of the CII of core router interfaces - total number of BW should at least reserved) /the number of queues should be created.

4.2 Policy Access Controller

Policy Access Controller is an apparatus for coordinating access of policy rules amongst a plurality of processes in a distributed system. It supports many read and one write locking mechanism. It also supports hierarchical locking, so that the clients of PAC have the flexibility to choose the

granularity of the lock.

1. The client creates a key according to the granularity of the object it wants to lock, it then issues a lock request to the lock server.

2. Upon the reception of the request, the lock server checks whether there is an existing lock in the write lock list that has a key that is the same as the requesting key? Or subset of the requesting key? Or super set of the requesting key?

3. If such a write lock exists, notify the client that someone has already used the key.

4. If no such a lock exist, and if the locking request is a read lock, place a new lock with the requesting key in the read lock list, grant the locking of the object. Return the success status to the client.

5. If no such a lock exist, and if the locking request is a write lock, check whether there is an existing lock in the read lock list that has a key is: the same as the requesting key? Or subset of the requesting key? Or super set of the requesting key?

6. If such a read lock exists, notify the client that someone has already used the key.

7. Otherwise, place a new lock with the requesting key in the write lock list, grant the locking of the object. Return the success status to the client.

One example of using PAC for the lock with no wait case

When a policy editor is trying to view the diffserv policy rules, PRA issues a read lock with no wait request to PAC on object key "PolicyGroupName = DiffServPolicy, qpDomainName = QoSTestBedDomain, dc = qospbn".

If there is an existing write lock with the key equals to one of the following, the lock request will be rejected

1. "qpDomainName = QoSTestBedDomain, dc = qospbn"

2. "PolicyGroupName = DiffServPolicy, qpDomainName = QoSTestBedDomain, dc = qospbn"

3. PolicyRuleName = SampleRule 1, PolicyGroupName = DiffServPolicy, qpDomainName = QoSTestBedDomain, dc = qospbn.

The request will be granted if the above listed keys are only belong to read locks, or no such keys exist at all.

Figure 8 shows the procedure of PAC for the lock with wait case.

1. The client creates a key according to the granularity of the object it want to lock, it then issue a lock request to the lock server.

2. Upon the reception of the request, the lock server checks whether there is an existing lock in the write lock list that has a key is: the same as the requesting key? Or subset of the requesting key? Or super set of the requesting key?

3. If such a write lock exists, put the key along with the callback object provided by the client into the waiting list, return the status to the client.

4. If no such a lock exist, and if the locking request is a read lock, place a new lock with the requesting key in the read lock list, grant the locking of the object. Return the success status to the client.

5. If no such a lock exist, and if the locking request is a write lock, check whether there is an existing lock in the read lock list that has a key is: the same as the requesting key? Or subset of the requesting key? Or super set of the requesting key?

6. If such a read lock exists, put the key along with the callback object provided by the client into the waiting list, return the status to the client.

7. Otherwise, place a new lock with the requesting key in the write lock list, grant the locking of the object. Return the success status to the client.

One example of using PAC for the lock with wait case

When a policy editor is trying to modify the diffserv policy rules, PRA issues a write lock with wait request to PAC on object key "PolicyRuleName = SampleRule1, PolicyGroupName=DiffServPolicy, qpDomamName=QoSTestBedDomain, dc=qospbn".

If there is an existing write, or read lock with the key

equals to one of the following, the lock request will be put into a waiting list, and the policy editor will be notified when the key has been released.

1. "qpDomainName=QoSTestBedDomain, dc=qospbn"

2. "PolicyGroupName=DiffServPolicy, qpDomainName=QoSTestBedDomain, dc=qospbn"

3. PolicyRuleName=SampleRule 1, PolicyGroupName=DiffServPolicy, qpDomainName=QoSTestBedDomain, dc=qospbn.

The request will be granted if the above listed keys are only belong to read locks, or no such keys exist at all.

Claims

1. Policy Management System comprising
- a first plurality of event generators,
- a second plurality of event consumers,
- at least one event server, and
- a communication mechanism coupling the event generators with the event consumers via the event server and providing event generation and reception.

2. System according to claim 1,
wherein the communication mechanism comprises a message oriented middleware, in particular CORBA.

3. System according to claim 1 or 2,
wherein in the event generators comprise
- a Policy Repository Adapter (PRA) for managing the access to a policy repository
- a Rule Scheduler scheduling the installation and de-installation of policy rules, and
- a Policy Enforcement Point (PEP) manager reporting events which are generated by the Policy Enforcement Points.

4. System according to one of the preceding claims,
wherein the event consumer comprise a Policy Decision Engine (PDE) managing the installation and de-installation of policy rules and receipt of events.

5. System according to one of the preceding claims
wherein the event generators are adapted to publish any set of events they are generating.

6. System according to one of the preceding claims,

wherein the event consumers are adapted to electronically subscribe to a predetermined set of events.

7. System according to one of claims 3 - 6,
wherein the Policy Enforcement Points manager is adapted to maintain a data-store containing
- information on those Policy Enforcement Points which are subject of the management and
- that management information which is maintained for each of the managed Policy Enforcement Points, in particular fault and performance data and configuration capability of the Policy Enforcement Point.

8. System according to claim 7,
wherein the Policy Enforcement Point manager is adapted to notify any changes to the data-store to the event server.

9. System according to one of claims 6 - 8,
wherein the Policy Enforcement Point manager is adapted to filter and forward evenst to an event consumer which subscribed to an predetermined set of events directly i.e. by passing the event server, and a Policy Decision Engine is provided which is adapted to execute the subscription of event directly from the Policy Enforcement Point manager.

10. System of claim 8 or 9, wherein the changes happening to the data store are reported as one of the following types of events:

I. Object creation event with object class name and other attributes of the object instance;

II. Object deletion event with the object instance name;

III. Attribute value change event with the object instance and attribute name, which indicate change in the existing value of attribute of an object;

IV. State change event which indicates a change in the fault state of a PEP or a particular entity in the PEP;

V. Threshold crossing event with the attribute that crossed the threshold and whether crossing is for the best or the worst

11. System according to one of claims 3 - 11, wherein changes happening to a policy rule in the policy repository are reported as events by the Policy Repository Adapter to the event server.

12. System according to one of claims 3 - 11, wherein the Policy Repository Adapter is capable of filtering and forwarding events to an event subscriber directly i.e., by-passing the event Server, the Policy Decision Engine can directly subscribe to events from the Policy Repository Adapter.

13. System according to claim 11 or 12, wherein the changes happening to the policy repository are reported as one of the following types of events:

I. Create event for a new policy rule;

II. Modify event for a modification of existing rule;

III. Delete event for a deletion of existing rule

14. System to one of the preceding claims comprising event structure that relates the event with policy rules.

15. A method of evaluating policy rules based on the event that is generated, comprising the following steps:

A. The event attributes are used to find the relevant rules that are to be retrieved for the event.

B. It is not required to retrieve policy rules if the event is a policy rule deletion event or if all the relevant rules are available in cache.

C. The relevant rules are retrieved from the policy repository using the attributes, operator and values available in the event as a query condition.

D. The rules are cached for later usage.

E. The rules are evaluated according to the conditions.

F. The targets are evaluated to identify the list of targets that rule will be applied.

G. The actions specified in the policy rule are converted to the methods provided by the managed object classes and managed object instances.

H. The managed object instances obtained in step G are deleted if it is a delete event or installed if it other types of events.

I. If the result is successful, the policy rule installation information is created maintaining the Policy Enforcement Point information of where the rule is installed if it is not a modify/delete event or the policy rule installation information is modified or deleted if it is a modify/delete event.

J. The outcome of policy rule execution is reported to a central logger.

16. Method according to claim 15,
wherein step (E) further comprises sub-steps to pass the rules whose the condition part consists of only the variables and values which are part of the event whereas for other rules, it marks that part of the condition to be true if the condition is satisfied, for later evaluation when the other parts of the condition are triggered by other events.

17. Method according to claim 15 or 16,
wherein step (F) further comprises sub-steps of evaluating Policy Enforcement Points where the rules need to be installed if the target is Policy Enforcement Point, wherein, if the events are either modify or delete of a policy rule from the policy repository, the targets are obtained from the policy rule installation information, whereas, if it is other types of events, the targets are obtained from the policy rule condition.

18. Method according to one of claims 15 - 17,
wherein the managed object instances referred in step (G) are obtained as follows: For a modify/delete event of a policy rule, only the managed object instances that are

installed for the rule being modified/deleted, is obtained from the policy rule installation information. For all other events, new object instances are instantiated.

19. Method according to one of claims 15 - 18, wherein steps (A - G) are carried out in the Policy Decision Engine and Local Policy Cache components whereas the steps (H - J) are carried out in the Policy Enforcement Point manager.

20. Method according to one of claims 15 - 19, wherein in an additional step connected to step (J) the Policy Enforcement Point manager can mark the set of provisioning objects creation commands with the policy rule name and the event that was responsible for its execution of the commands from the Policy Decision Engine and this is used for processing the same event from some other Policy Enforcement Point at a later time.

21. Method according to one of claims 15 - 20, wherein in recourse of an interaction between various components when creating a new policy rule the following steps are carried out:

A. Policy Repository Adapter is called when a new rule is going to be created.

B. Policy Repository Adapter sends the policy rule to Policy Consistency Checker for conflict checking.

C. Policy Consistency Checker gets the policy rules from repository only when it is necessary.

D. Policy Consistency Checker uses Policy Access Controller to control the access to policy rules.

E. Policy Consistency Checkers finds out the conflicts and provides a list of possible solution.

F. Policy Repository Adapter provides an operation for the client to update the suggested policy rules to solve the conflict.

G. Policy Consistency Checker checks whether the updated policy rules conformed with the suggestions given.

H. Policy Repository Adapter updates the policy rules when no conflicts are found and release the lock in Policy Access Controller.

I. Policy Repository Adapter pushes an policy rule created event to event server.

22. A method of policy rule consistency checking which uses the following information:

I. the relationship information between object classes defined in the object model schema used for storing object instances in the Policy Enforcement Point manager data store; «example is filter output has to be input of queue»

II. other relationship information that can not be specified in the object model schema «add example for this which is queue bandwidth in the description» and

III. the actual instantiation of the object classes in the

object model which is maintained by the Policy Enforcement Point manager in the data-store «topology information is one example of this».

23. Method according to claim 22 comprises the following steps:

A. The schema for the object model is specified in a managed object class definition language

B. Managed Object Format (MOF) can be one example of the managed object class definition language.

C. The user specifies the constraints of second type identified in claim 22 using a constraint specification language

D. Object Constraint Language (OCL) can be one example of the constraint specification language.

E. As one of the inputs to the consistency checking, the actual instantiation of the object classes in the object model which reflects the network being managed, is processed.

F. The policy rules related to the new rule to be created are retrieved using presence of one or more attribute names or object classes used in the new rule to be created.

G. The condition of the new rule that is to be created is compared with the conditions of the retrieved rules for overlaps, the actions of the existing rule is compared against the actions of the new rule for conflicts.

H. If the result of evaluation is negative, the actions of all the existing rules are compared against the actions of the new rule for conflicts using simulation.

I. For all the conflicting rules found, the rule priorities are modified for resolving the conflict.

J. The information of the network nodes can be used for detecting the conflicts.

24. A method of controlling access to hierarchical information storage by creating read or write locks at various levels of the hierarchy, comprising the following steps:

A. The entity accessing the hierarchical storage generates the key for specifying the level at which locking is to be done.

B. Check is done for existence of write-lock and if present, the lock request is rejected and the method ends.

C. Check if the request is a read-lock and add the new lock to the list of read-locks pending if no existence of write lock is found.

D. Accept the lock request and add it to pending write lock list if the requested lock is not a read lock and there are no read locks pending.

25. Use of the method according to claim 24 for controlling concurrent accesses to the hierarchical storage of policy rules in a directory.

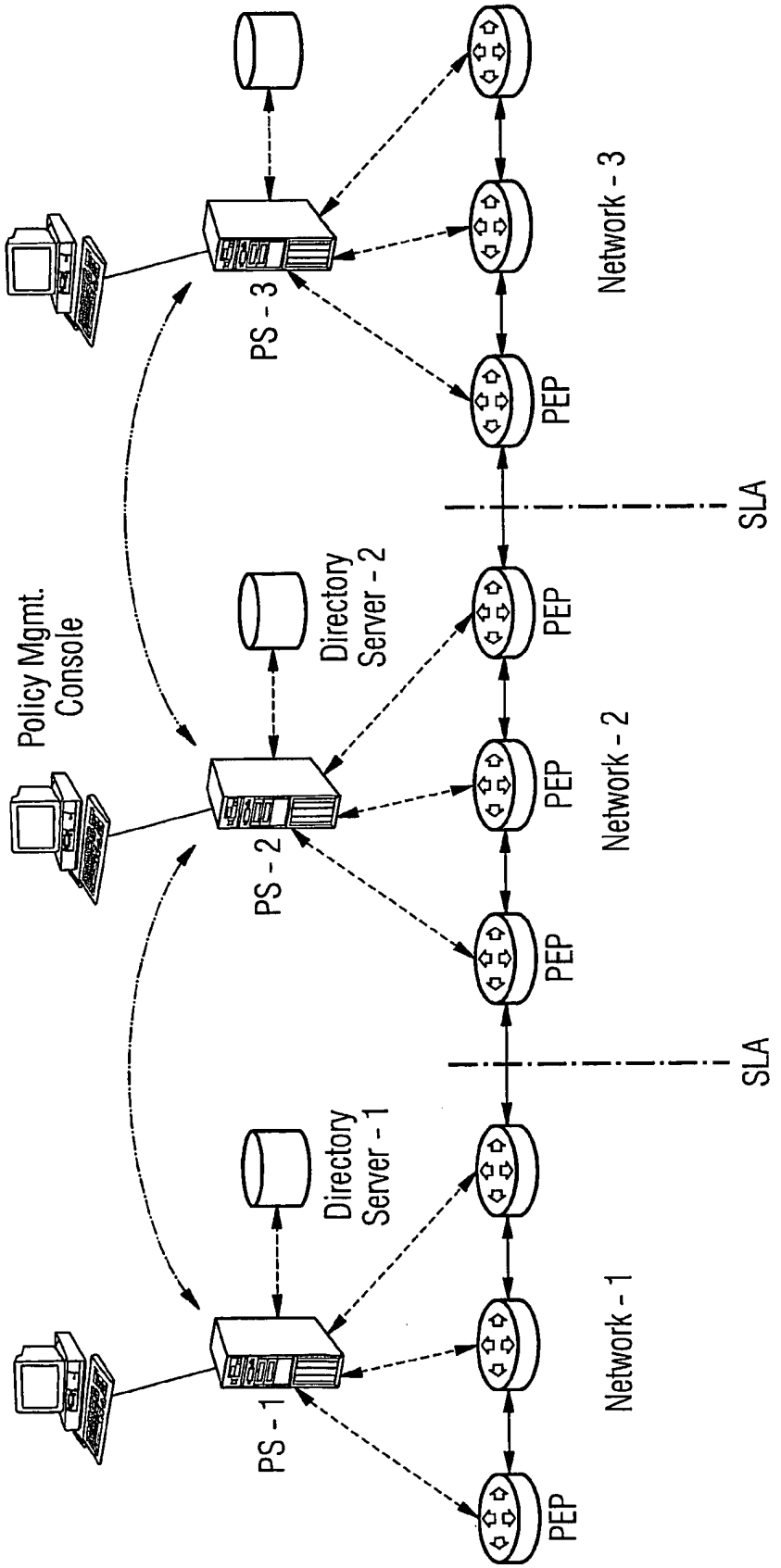

Policy Mgmt. Console
PS - 3
PS - 2
PS - 1
Directory Server - 2
Directory Server - 1
PEP
PEP
PEP
PEP
PEP
PEP
Network - 3
Network - 2
Network - 1
SLA
SLA

FIG 1

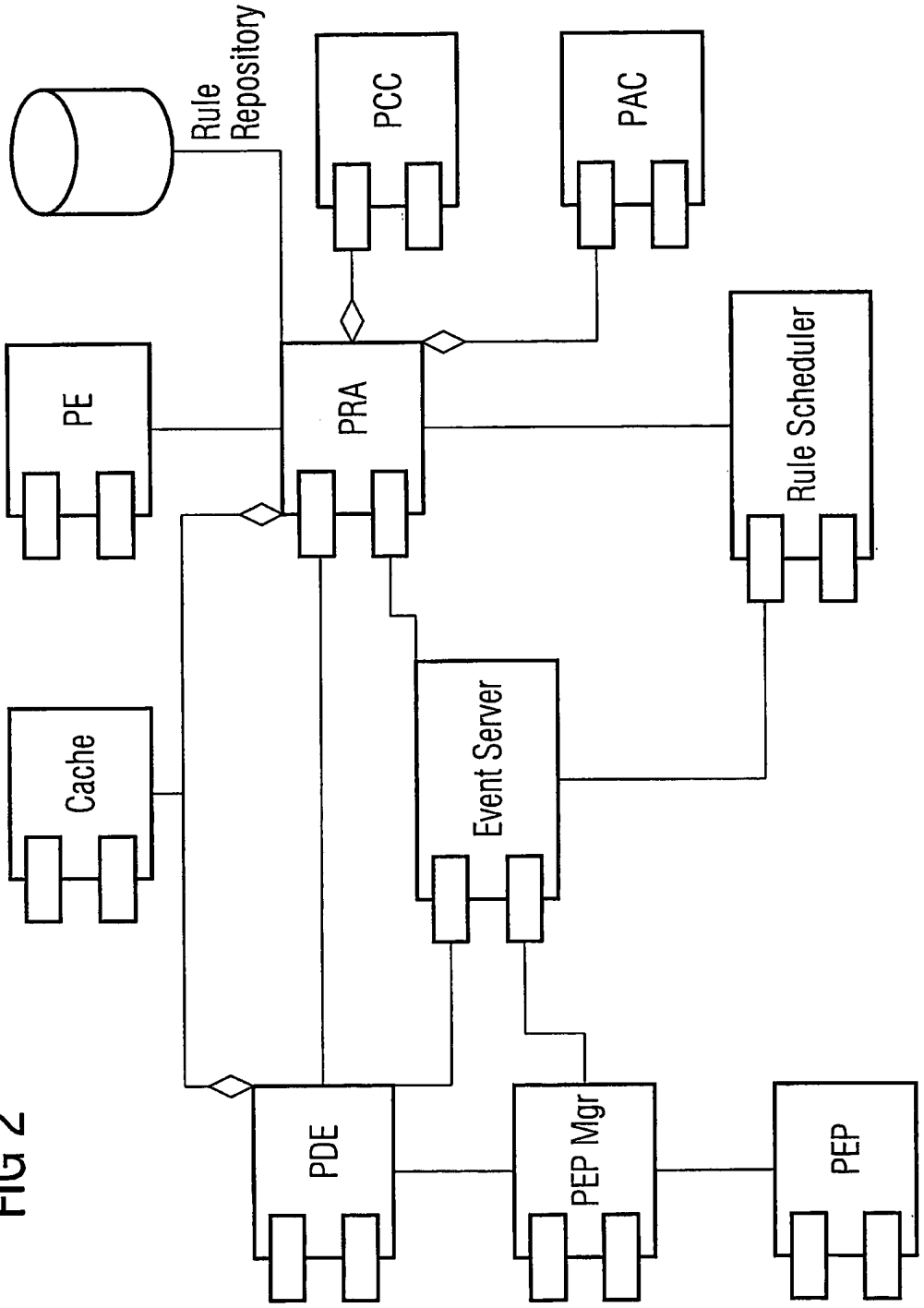

Rule Repository
PCC
PAC
PE
PRA
Rule Scheduler
Cache
Event Server
PDE
PEP Mgr
PEP

FIG 2

FIG 3

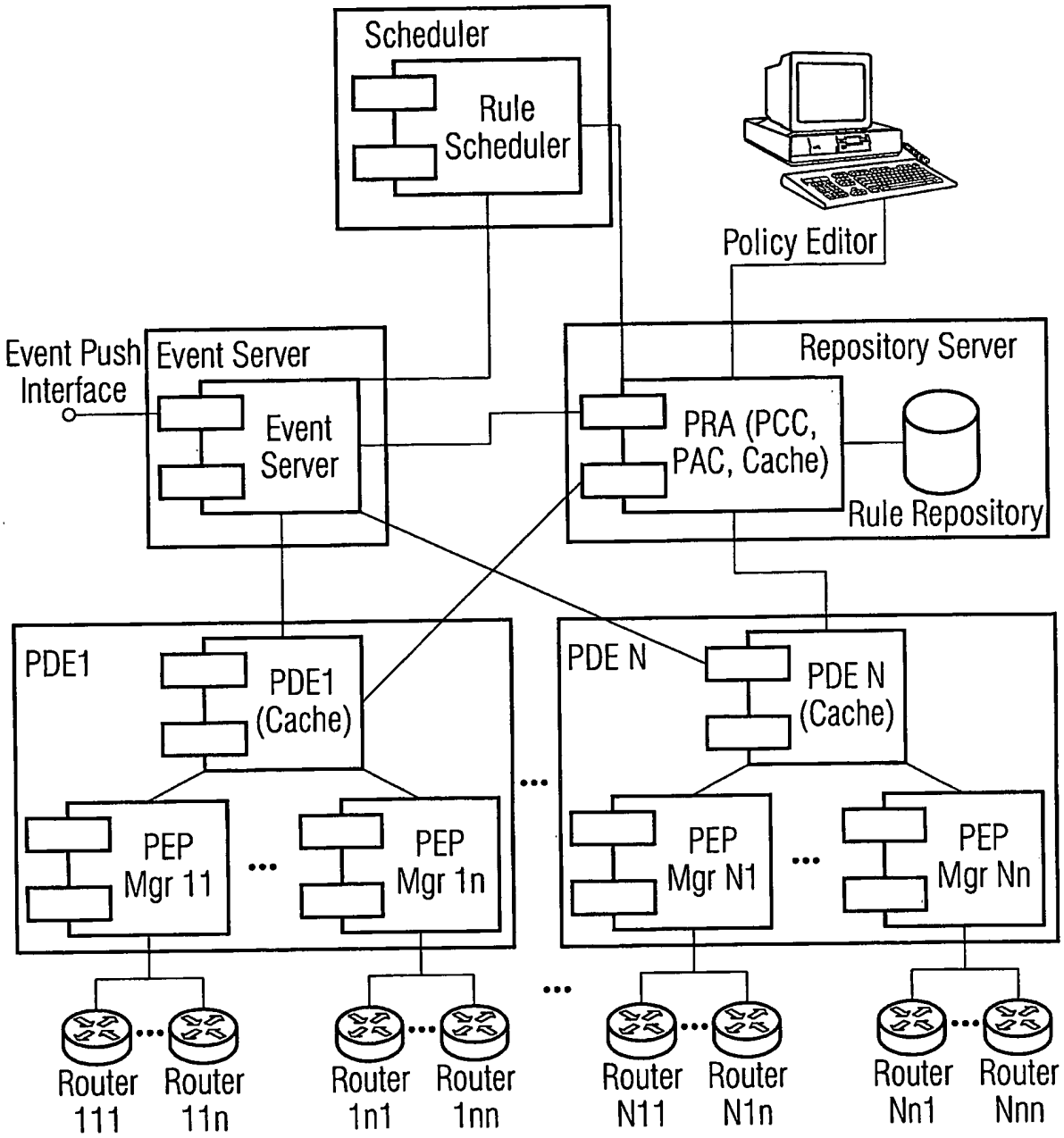

Scheduler
Rule Scheduler
Policy Editor
Event Push Interface
Event Server
Event Server
Repository Server
PRA (PCC, PAC, Cache)
Rule Repository
PDE1
PDE1 (Cache)
PDE N
PDE N (Cache)
PEP Mgr 11
PEP Mgr 1n
PEP Mgr N1
PEP Mgr Nn
Router 111
Router 11n
Router 1n1
Router 1nn
Router N11
Router N1n
Router Nn1
Router Nnn

FIG 4

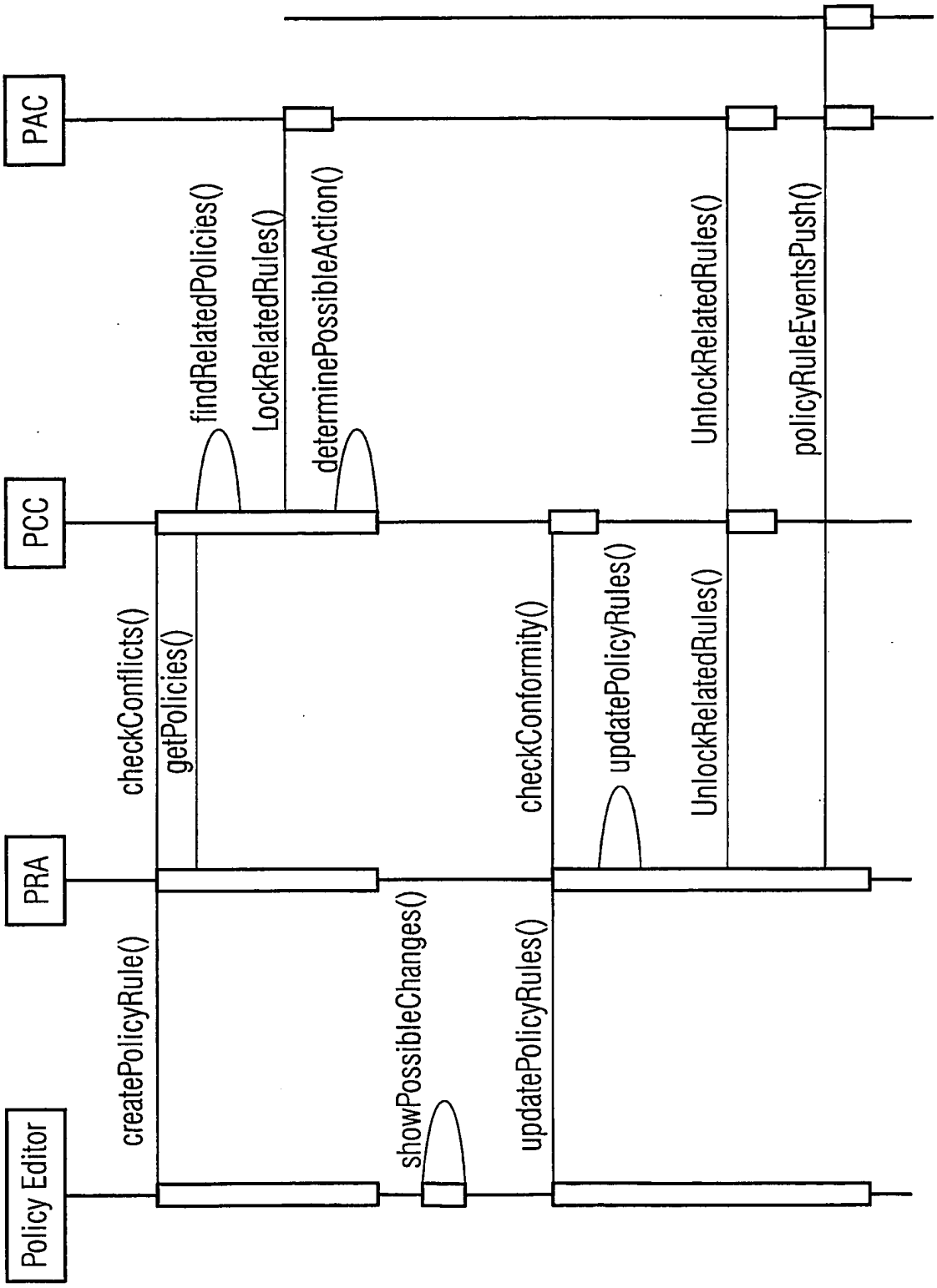

PAC
PCC
PRA
Policy Editor
findRelatedPolicies()
LockRelatedRules()
determinePossibleAction()
UnlockRelatedRules()
policyRuleEventsPush()
checkConflicts()
getPolicies()
checkConformity()
updatePolicyRules()
UnlockRelatedRules()
createPolicyRule()
showPossibleChanges()
updatePolicyRules()

FIG 5

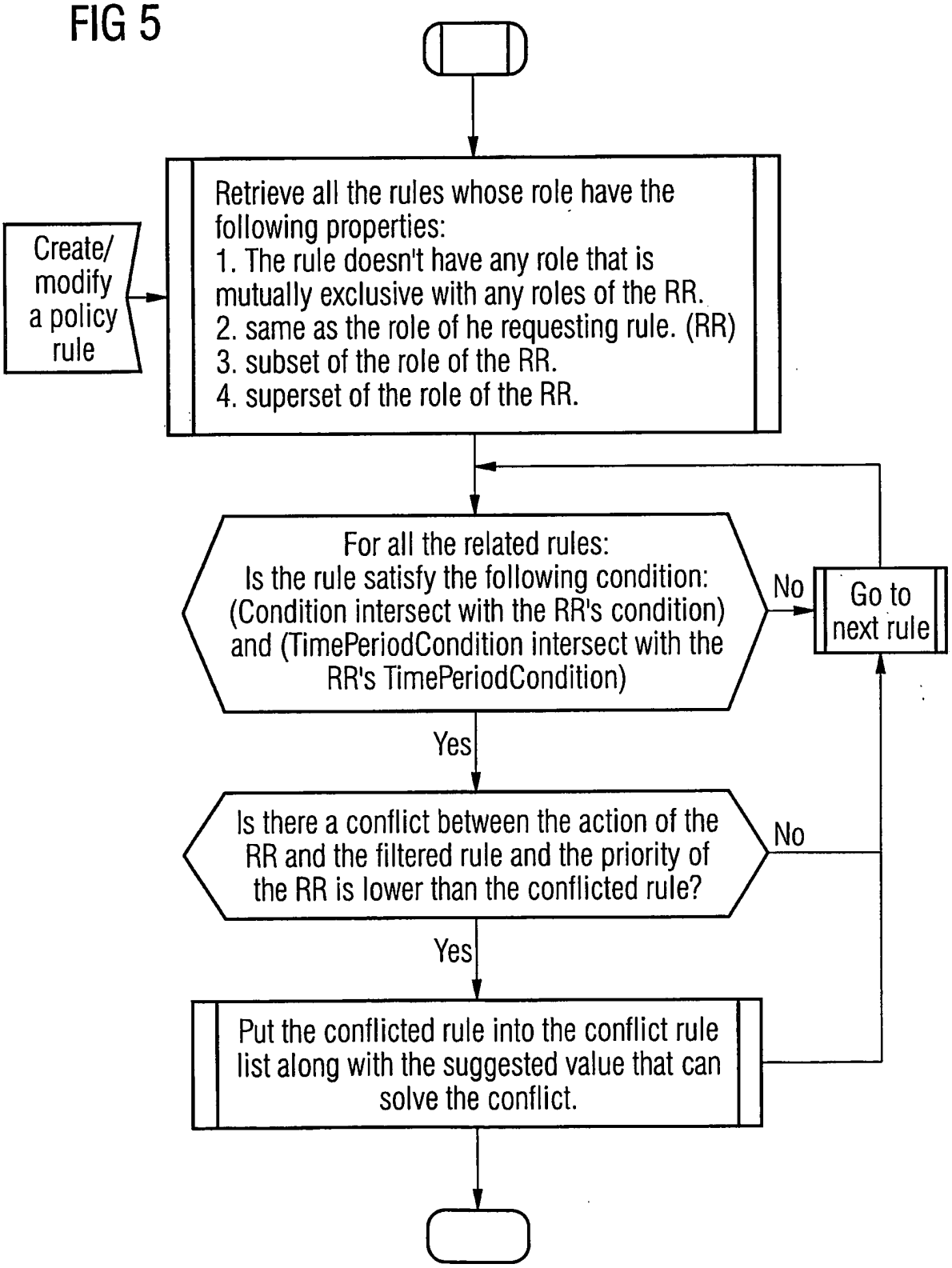

Create/ modify a policy rule
Retrieve all the rules whose role have the following properties:
1. The rule doesn't have any role that is mutually exclusive with any roles of the RR.
2. same as the role of he requesting rule. (RR)
3. subset of the role of the RR.
4. superset of the role of the RR.
For all the related rules:
Is the rule satisfy the following condition:
(Condition intersect with the RR's condition) and (TimePeriodCondition intersect with the RR's TimePeriodCondition)
No
Go to next rule
Yes
Is there a conflict between the action of the RR and the filtered rule and the priority of the RR is lower than the conflicted rule?
No
Yes
Put the conflicted rule into the conflict rule list along with the suggested value that can solve the conflict.

FIG 6

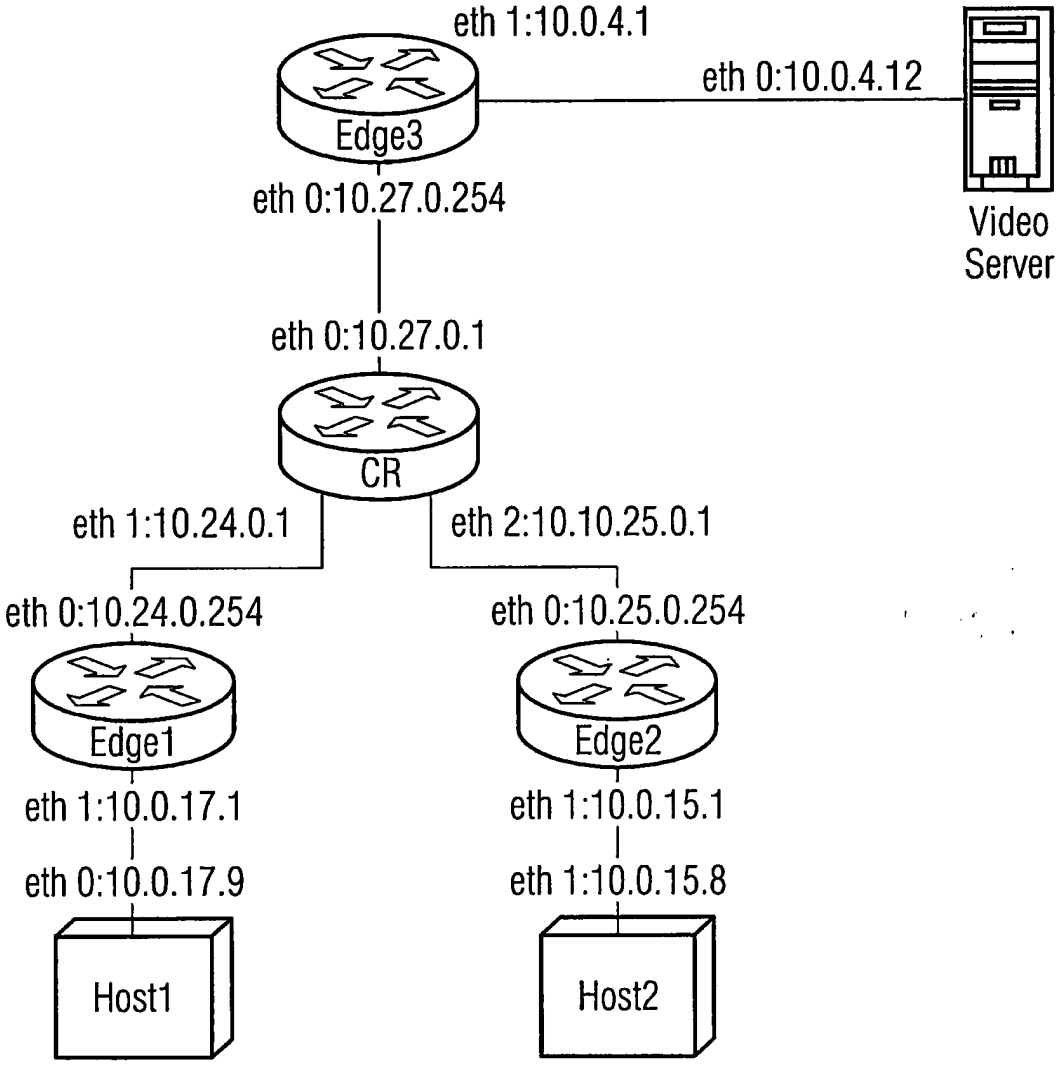

eth 1:10.0.4.1
eth 0:10.0.4.12
Edge3
Video Server
eth 0:10.27.0.254
eth 0:10.27.0.1
CR
eth 1:10.24.0.1
eth 2:10.10.25.0.1
eth 0:10.24.0.254
eth 0:10.25.0.254
Edge1
Edge2
eth 1:10.0.17.1
eth 1:10.0.15.1
eth 0:10.0.17.9
eth 1:10.0.15.8
Host1
Host2

FIG 7

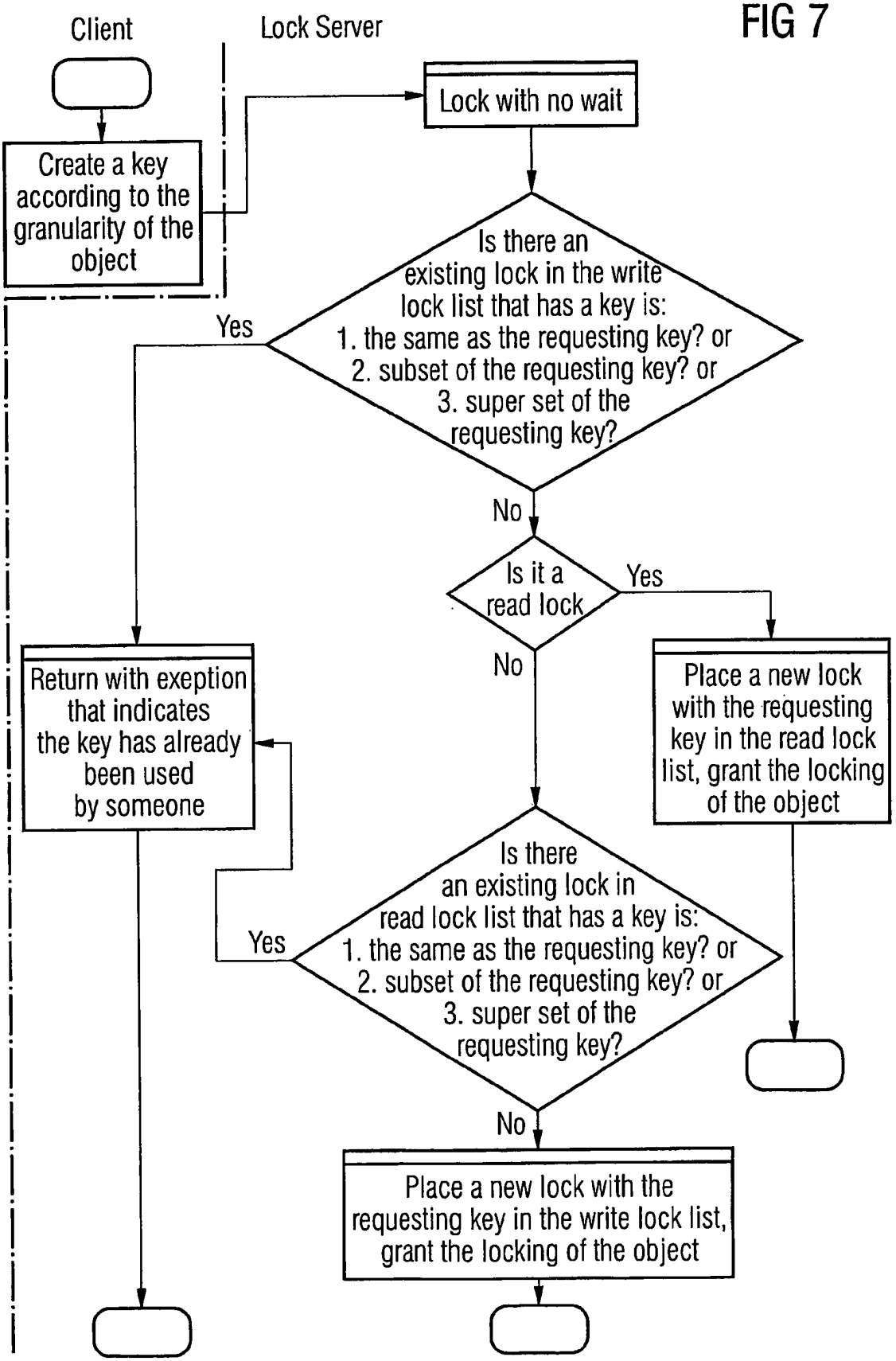
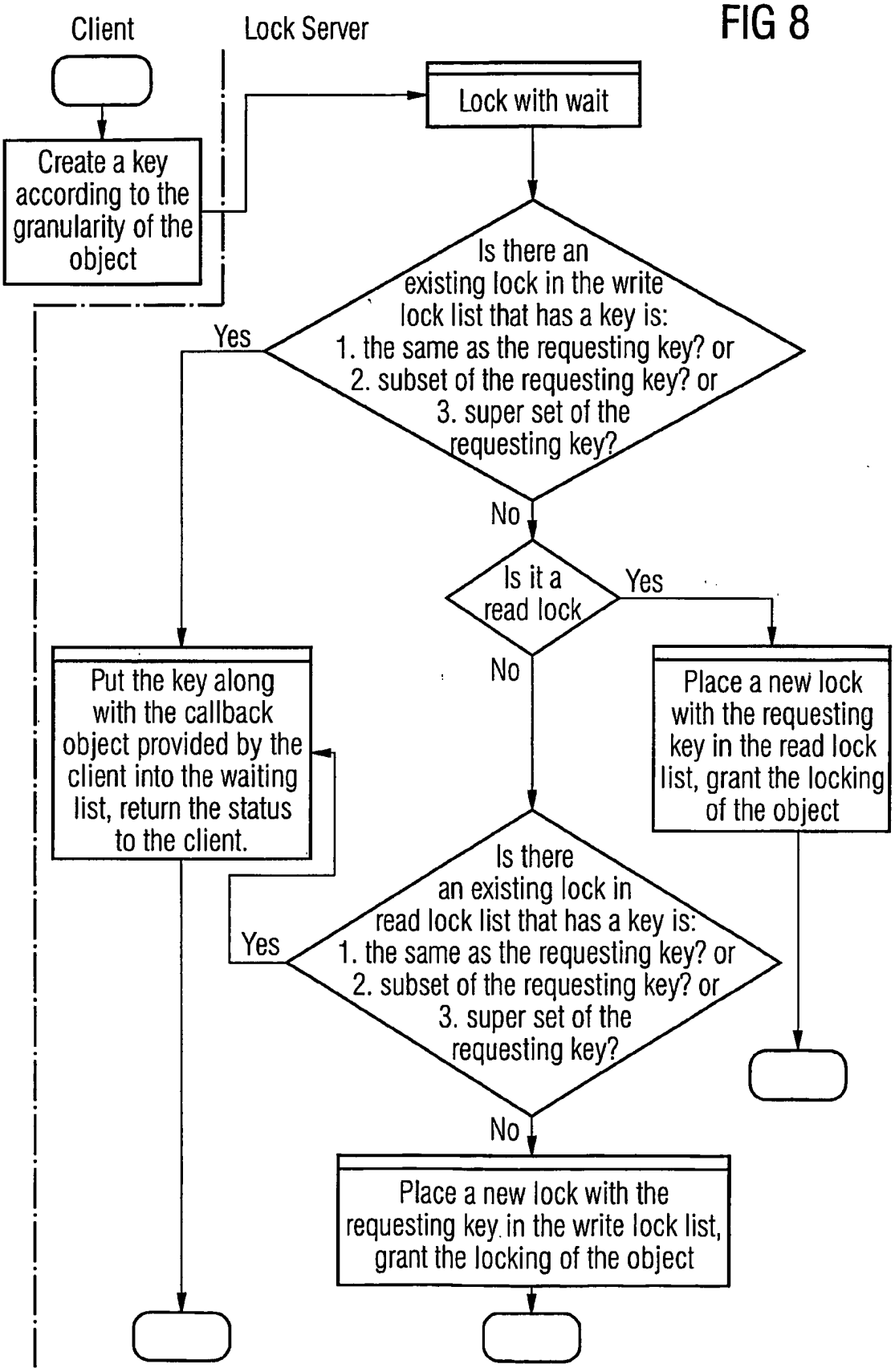

Client
Lock Server
Create a key according to the granularity of the object
Lock with no wait
Is there an existing lock in the write lock list that has a key is:
1. the same as the requesting key? or
2. subset of the requesting key? or
3. super set of the requesting key?
Yes
No
Is it a read lock
Yes
No
Return with exeption that indicates the key has already been used by someone
Place a new lock with the requesting key in the read lock list, grant the locking of the object
Is there an existing lock in read lock list that has a key is:
1. the same as the requesting key? or
2. subset of the requesting key? or
3. super set of the requesting key?
Yes
No
Place a new lock with the requesting key in the write lock list, grant the locking of the object

FIG 8


Client
Lock Server
Create a key according to the granularity of the object
Lock with wait
Is there an existing lock in the write lock list that has a key is:
1. the same as the requesting key? or
2. subset of the requesting key? or
3. super set of the requesting key?
Yes
No
Is it a read lock
Yes
No
Put the key along with the callback object provided by the client into the waiting list, return the status to the client.
Place a new lock with the requesting key in the read lock list, grant the locking of the object
Is there an existing lock in read lock list that has a key is:
1. the same as the requesting key? or
2. subset of the requesting key? or
3. super set of the requesting key?
Yes
No
Place a new lock with the requesting key in the write lock list, grant the locking of the object

INTERNATIONAL SEARCH REPORT

International Application No
PCT/EP 02/06975

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L12/24

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 1 026 867 A (NORTEL NETWORKS CORP) 9 August 2000 (2000-08-09) paragraphs [0045],[0053],[0060],[0064],[0065],[0068], [0079],[0080],[0101],[0103],[0104],[0106] figures 2,3,5 ---	1,3-14
X	US 2002/050926 A1 (DATTA UTPAL ET AL) 2 May 2002 (2002-05-02) paragraphs [0078],[0090] figures 2,8,12 --- -/--	1,3-14

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
3 April 2003	17 07 2003

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Siebel, C

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/EP 02/06975

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

1-14

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. Claims: 1-14

   A policy management system comprising a communication system using a message oriented middleware (e.g. Corba)

2. Claims: 15-21

   A method for evaluating policy rules

3. Claims: 22-23

   A method of policy consistency checking

4. Claims: 24-25

   A method of controlling access to hierarchical information storage

INTERNATIONAL SEARCH REPORT

International Application No
PCT/EP 02/06975

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	REARDON M: "MAKING POLICY IS AS EASY AS PLUG AND PLAY" DATA COMMUNICATIONS, MCGRAW HILL. NEW YORK, US, vol. 28, no. 10, July 1999 (1999-07), pages 29-30, XP000846704 ISSN: 0363-6399 page 29 figure 1 ---	2

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/EP 02/06975

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 1026867 A	09-08-2000	EP 1026867 A2	09-08-2000
US 2002050926 A1	02-05-2002	US 6255943 B1	03-07-2001
		US 6057757 A	02-05-2000
		US 5777549 A	07-07-1998
		US 6373383 B1	16-04-2002
		AT 219874 T	15-07-2002
		AU 720061 B2	25-05-2000
		AU 5325896 A	16-10-1996
		DE 69622026 D1	01-08-2002
		DE 69622026 T2	27-02-2003
		EP 0818096 A1	14-01-1998
		WO 9631035 A1	03-10-1996
		US 5696486 A	09-12-1997